



Robert D. Falgout
Center for Applied Scientific Computing

ACTS Toolkit Workshop
September 28-30, 2000



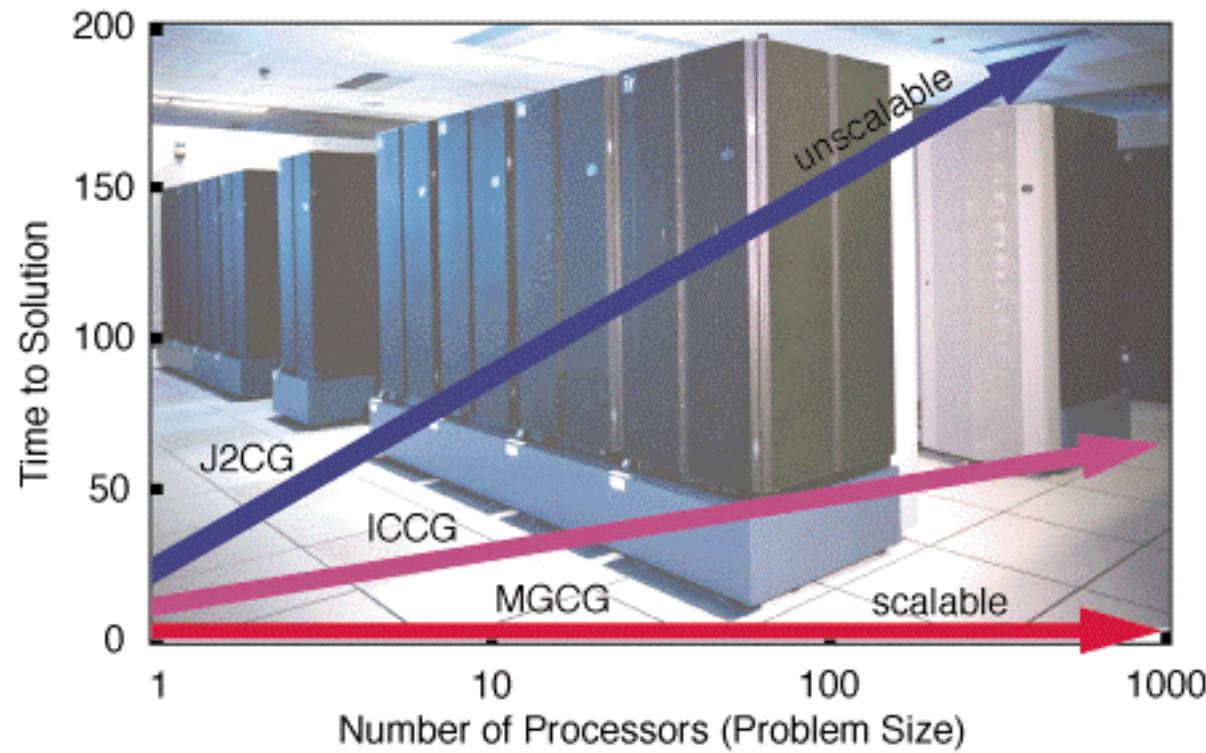
Outline

- **Introduction / Motivation**
- **Getting Started / Conceptual Interfaces**

- **Structured-Grid Interface (`struct`)**
- **Semi-Structured-Grid Interface (`sstruct`)**
- **Finite Element Interface (`FEI`)**
- **Linear-Algebraic Interface (`IJ`)**

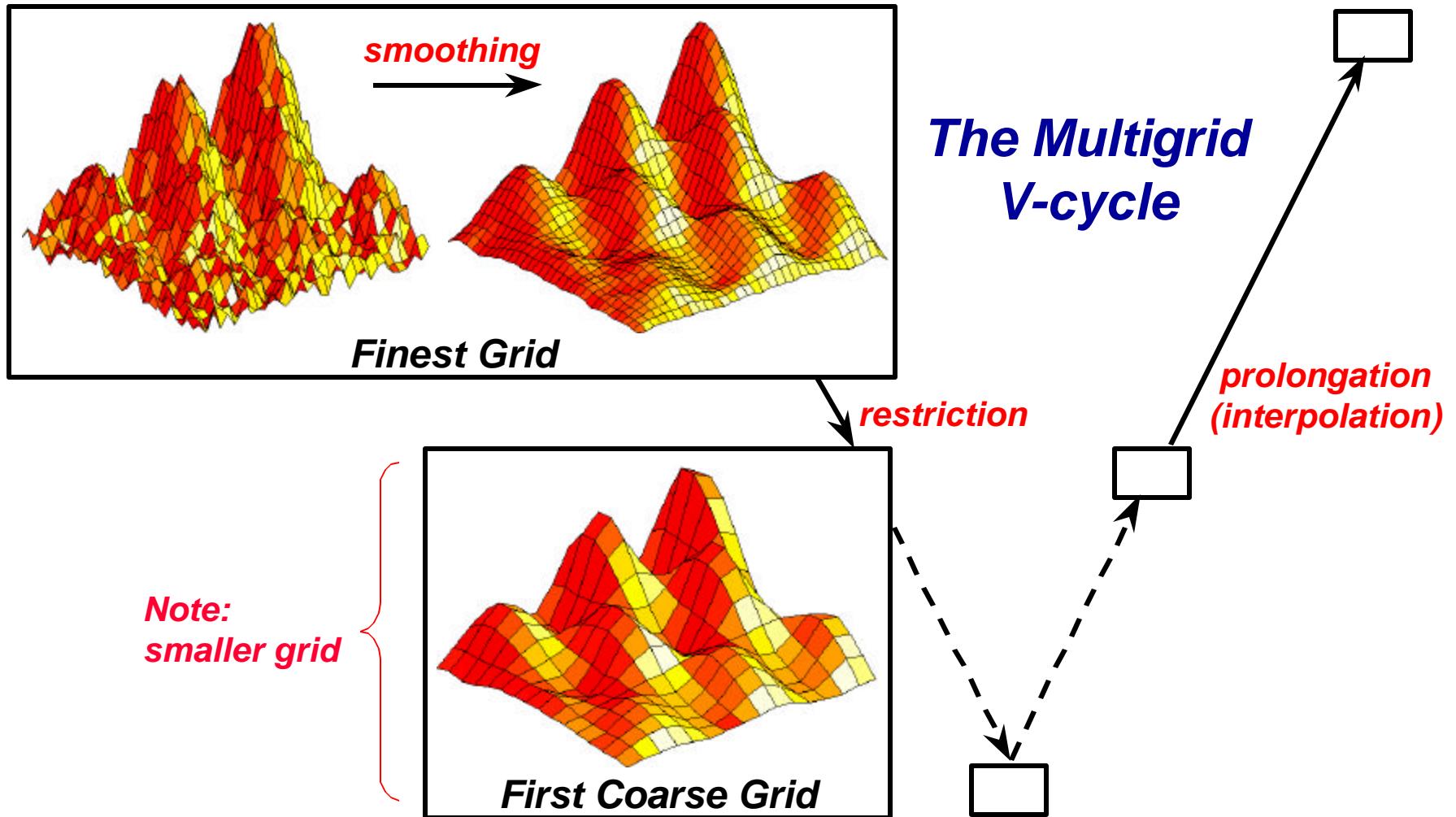
- **Solvers and Preconditioners**
- **Additional Information**

Scalability is a central issue for large-scale parallel computing



Linear solver convergence can be discussed independent of parallel computing, and is often overlooked as a key scalability issue.

Multigrid uses coarse grids to efficiently damp out smooth error components

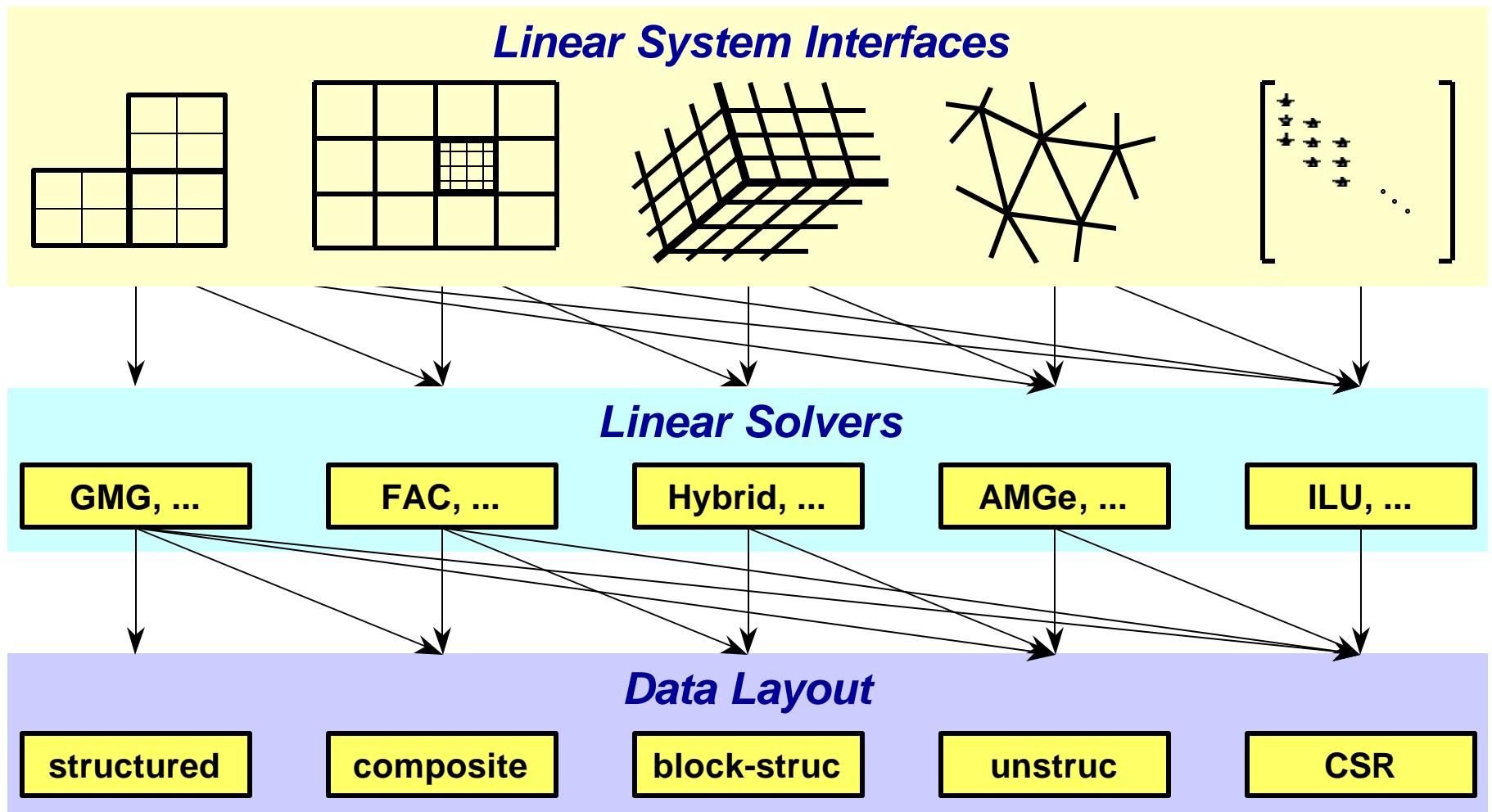


Getting Started

- Before writing your code:
 - choose a conceptual interface
 - choose a solver / preconditioner
 - choose a matrix type that is compatible with your solver / preconditioner and conceptual interface

- Now write your code:
 - build auxiliary structures (e.g., grids, stencils)
 - build matrix/vector through conceptual interface
 - build solver/preconditioner
 - solve the system
 - get desired information from the solver

Multiple interfaces are necessary to provide “best” solvers and data layouts



Why multiple interfaces? The key points

- Provides natural “views” of the linear system
- Eases some of the coding burden for users by eliminating the need to map to rows/columns
- Provides for more efficient (scalable) linear solvers
- Provides for more effective data storage schemes and more efficient computational kernels

hypre currently supports four conceptual interfaces

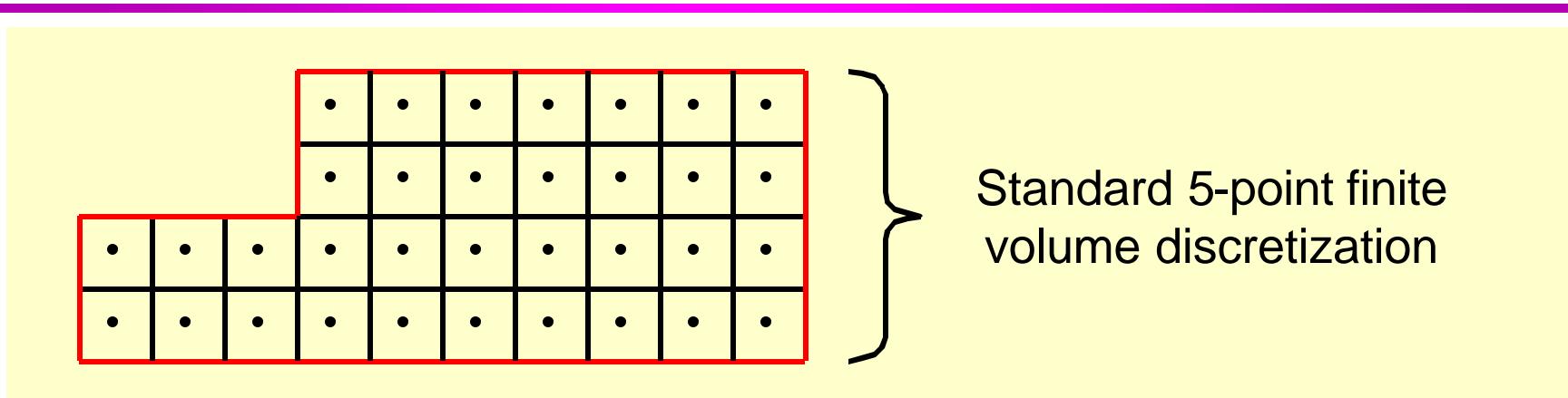
- **Structured-Grid Interface ([Struct](#))**
 - *applications with logically rectangular grids*
- **Semi-Structured-Grid Interface ([Sstruct](#))**
 - *applications with grids that are mostly structured, but with some unstructured features (e.g., block-structured, AMR, overset)*
- **Finite Element Interface ([FEI](#))**
 - *unstructured-grid, finite element applications*
- **Linear-Algebraic Interface ([IJ](#))**
 - *applications with sparse linear systems*
- **More about each next...**

Structured-Grid System Interface (Struct)

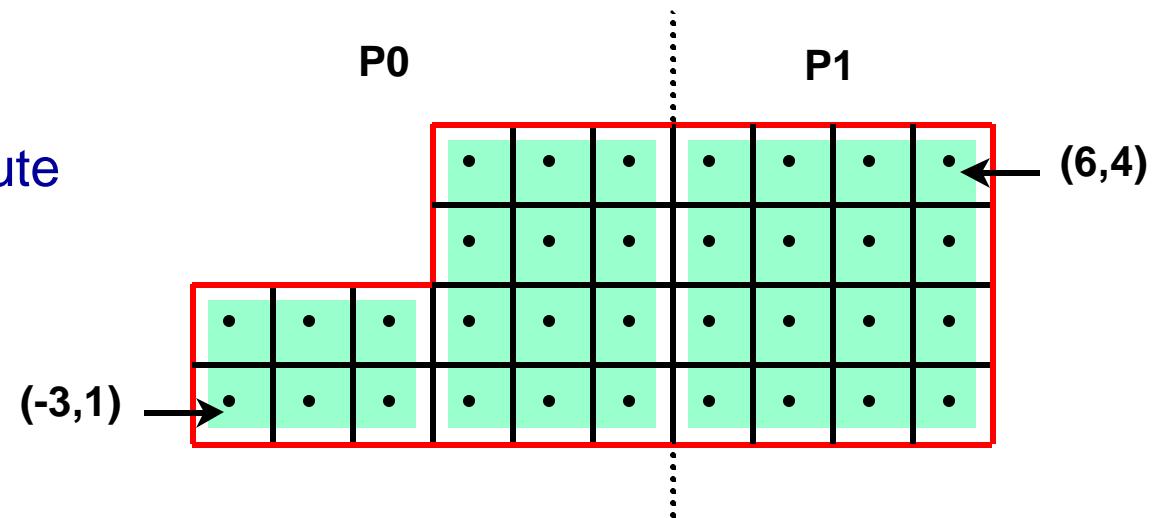
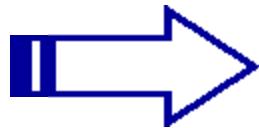
- There are four basic steps involved:
 - set up the Grid
 - set up the stencil
 - set up the Matrix
 - set up the right-hand-side vector
- Consider the following 2D Laplacian problem

$$\begin{cases} \nabla^2 u = f, & \text{in the domain} \\ u = 0, & \text{on the boundary} \end{cases}$$

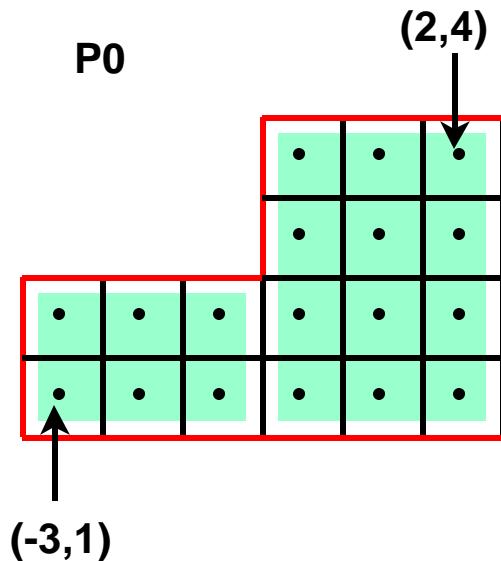
A structured-grid finite volume example:



Partition and distribute



A structured-grid finite volume example : *setting up the Grid*



```
HYPRE_StructGrid grid;
int ilo[2][2] = {{-3, 1}, {0, 1}};
int iup[2][2] = {{-1, 2}, {2, 4}};

HYPRE_StructGridCreate(MPI_COMM_WORLD,
                      2, &grid);

HYPRE_StructGridSetExtents(grid,
                           ilo[0], iup[0]);
HYPRE_StructGridSetExtents(grid,
                           ilo[1], iup[1]);

HYPRE_StructGridAssemble(grid);
```

A structured-grid finite volume example : *setting up the stencil*

$$\begin{bmatrix} & (0,1) \\ (-1,0) & (0,0) & (1,0) \\ & (0,-1) \end{bmatrix}$$



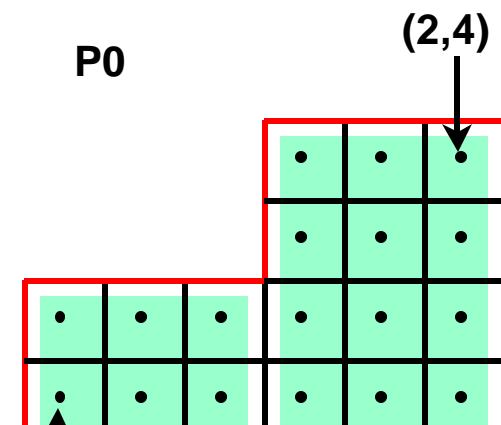
$$\begin{bmatrix} & S_4 \\ S_1 & S_0 & S_2 \\ & S_3 \end{bmatrix}$$

```
HYPRE_StructStencil stencil;
int offsets[5][2] = {{0,0},
                     {-1,0}, {1,0},
                     {0,-1}, {0,1}};
int s;

HYPRE_StructStencilCreate(2, 5,
                         &stencil);

for (s = 0; s < 5; s++)
{
    HYPRE_StructStencilSetElement(
        stencil, s, offsets[s]);
}
```

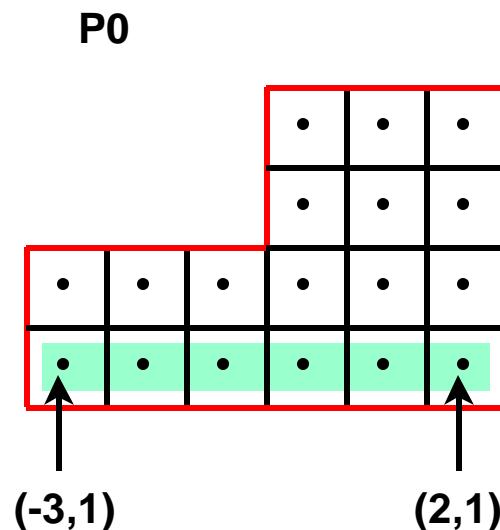
A structured-grid finite volume example : *setting up the Matrix*



$$\begin{bmatrix} S4 \\ S1 \ S0 \ S2 \\ S3 \end{bmatrix} = \begin{bmatrix} -1 & & \\ -1 & 4 & -1 \\ & -1 \end{bmatrix}$$

```
HYPRE_StructMatrix A;  
double values[36] = {4, -1, 4, -1, ...};  
int entries[2] = {0,3};  
  
HYPRE_StructMatrixCreate(MPI_COMM_WORLD,  
    grid, stencil, &A);  
HYPRE_StructMatrixInitialize(A);  
  
HYPRE_StructMatrixSetBoxValues(A,  
    ilo[0], iup[0], 2, entries, values);  
HYPRE_StructMatrixSetBoxValues(A,  
    ilo[1], iup[1], 2, entries, values);  
  
/* set boundary conditions */  
...  
  
HYPRE_StructMatrixAssemble(A);
```

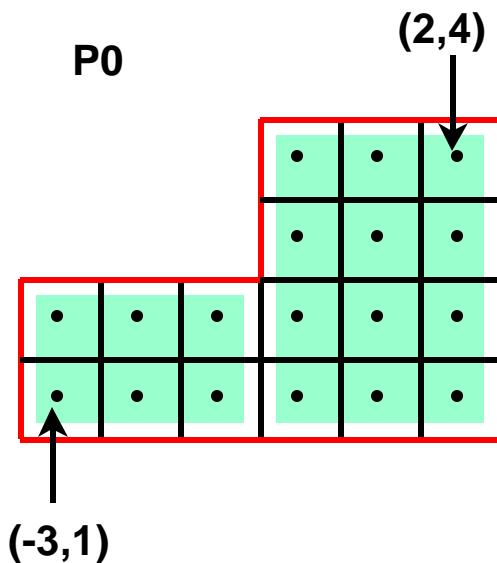
A structured-grid finite volume example : *setting up the Matrix (bc's)*



$$\begin{pmatrix} S_4 \\ S_1 & S_0 & S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 & 4 & -1 \\ 0 \end{pmatrix}$$

```
int    ilo[2] = {-3, 1};  
int    iup[2] = { 2, 1};  
double values[12] = {0, 0, ...};  
  
/* set interior coefficients */  
...  
  
/* implement boundary conditions */  
...  
  
i = 3;  
HYPRE_StructMatrixSetBoxValues(A,  
                                ilo, iup, 1, &i, values);  
  
/* complete implementation of bc's */  
...
```

A structured-grid finite volume example : *setting up the right-hand-side vector*



```
HYPRE_StructVector b;  
double values[18] = {0, 0, ...};  
  
HYPRE_StructVectorCreate(MPI_COMM_WORLD,  
    grid, &b);  
HYPRE_StructVectorInitialize(b);  
  
HYPRE_StructVectorSetBoxValues(b,  
    ilo[0], iup[0], values);  
HYPRE_StructVectorSetBoxValues(b,  
    ilo[1], iup[1], values);  
  
HYPRE_StructVectorAssemble(b);
```

Symmetric Matrices

- Some solvers support symmetric storage
- Between `Create()` and `Initialize()`, call:

```
HYPRE_StructMatrixSetSymmetric(A, 1);
```

- For best efficiency, only set half of the coefficients

$$\begin{bmatrix} & (0,1) \\ (0,0) & (1,0) \end{bmatrix} \iff \begin{bmatrix} s_2 \\ s_0 & s_1 \end{bmatrix}$$

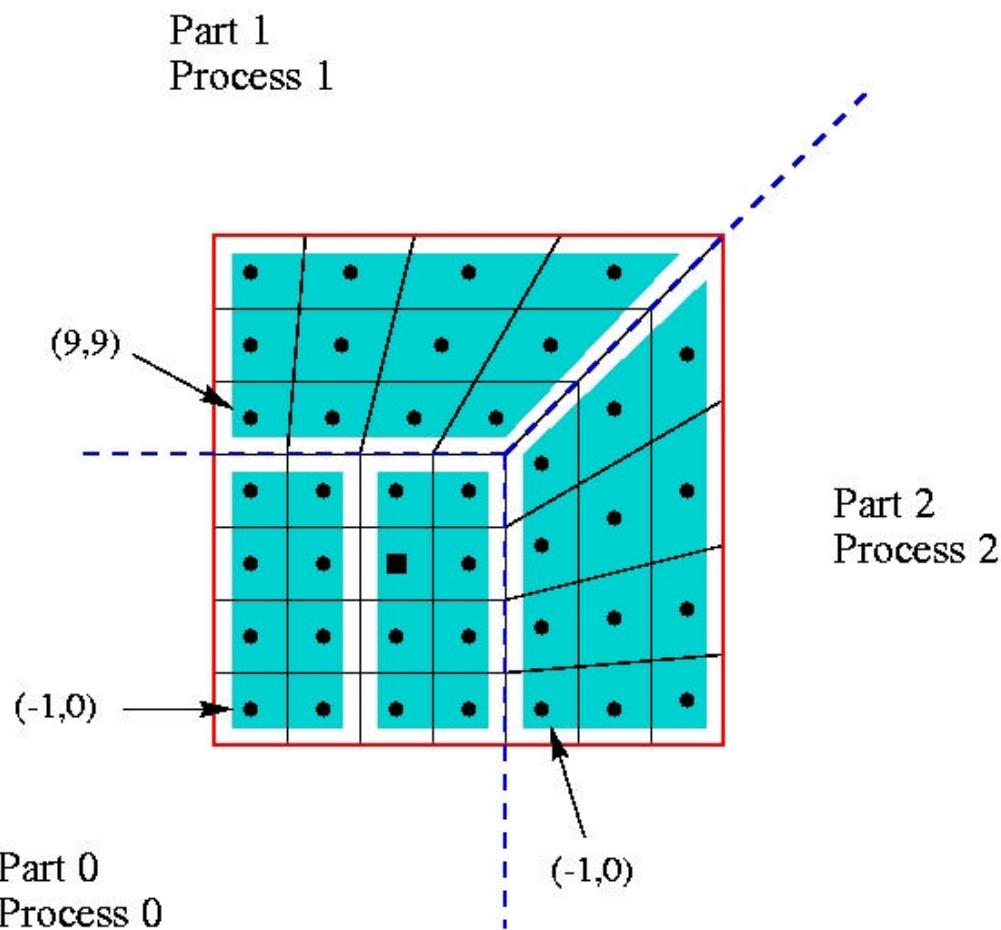
- This is enough info to recover the full 5-pt stencil

Semi-Structured-Grid System Interface (Sstruct - beta code)

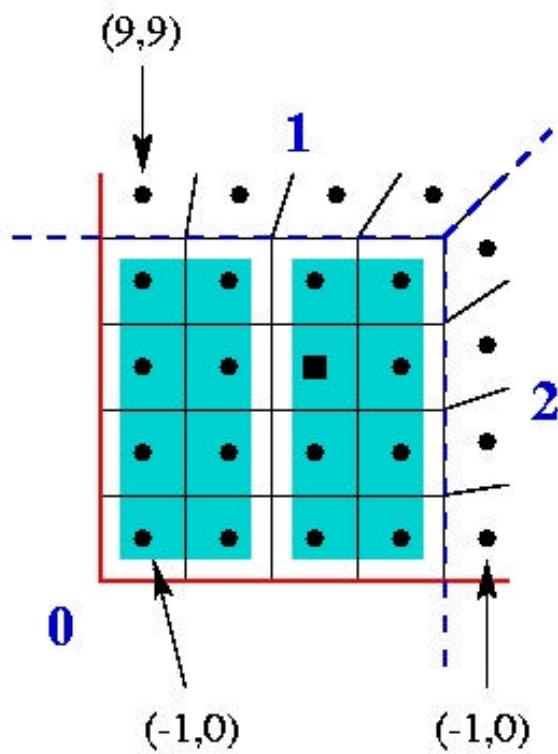
- There are five basic steps involved:
 - set up the Grid
 - set up the Stencils
 - set up the Graph
 - set up the Matrix
 - set up the right-hand-side vector
- Consider again the following 2D Laplacian problem

$$\begin{cases} \nabla^2 u = f, & \text{in the domain} \\ u = 0, & \text{on the boundary} \end{cases}$$

A block-structured grid example



A block-structured grid example : *setting up the Grid*



```
HYPRE_SStructVariable vars[1] =  
    {HYPRE_SSTRUCT_VARIABLE_CELL};  
int addindex[2] = {1,2};  
HYPRE_SStructVariable addvars[1] =  
    {HYPRE_SSTRUCT_VARIABLE_CELL};  
  
HYPRE_SStructGridCreate(MPI_COMM_WORLD,  
    2, 3, &grid);  
  
HYPRE_SStructGridSetExtents(grid, 0,  
    ilo[0], iup[0]);  
...  
HYPRE_SStructGridSetVariables(grid, 0,  
    1, vars);  
HYPRE_SStructGridAddVariables(grid, 0,  
    addindex, 1, addvars);  
  
HYPRE_SStructGridAssemble(grid);
```

A block-structured grid example : *setting up the stencil*

$$\begin{bmatrix} (-1,1) & (0,1) & (1,1) \\ (-1,0) & (0,0) & (1,0) \\ (-1,-1) & (0,-1) & (1,-1) \end{bmatrix}$$



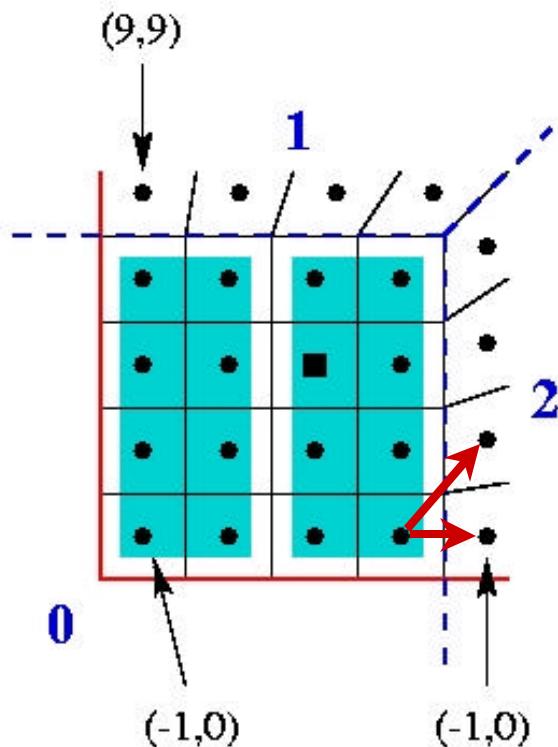
$$\begin{bmatrix} S7 & S4 & S8 \\ S1 & S0 & S2 \\ S5 & S3 & S6 \end{bmatrix}$$

```
HYPRE_SStructStencil stencil;
int s;
int offsets[9][2] = {{0,0},
                     {-1, 0}, { 1, 0},
                     { 0,-1}, { 0, 1},
                     {-1,-1}, { 1,-1},
                     {-1, 1}, { 1, 1}};

HYPRE_SStructStencilCreate(2, 9,
                           &stencil);

for (s = 0; s < 9; s++)
{
    HYPRE_SStructStencilSetEntry(stencil,
                                 s, offsets[s], 0);
}
```

A block-structured grid example : *setting up the Graph*



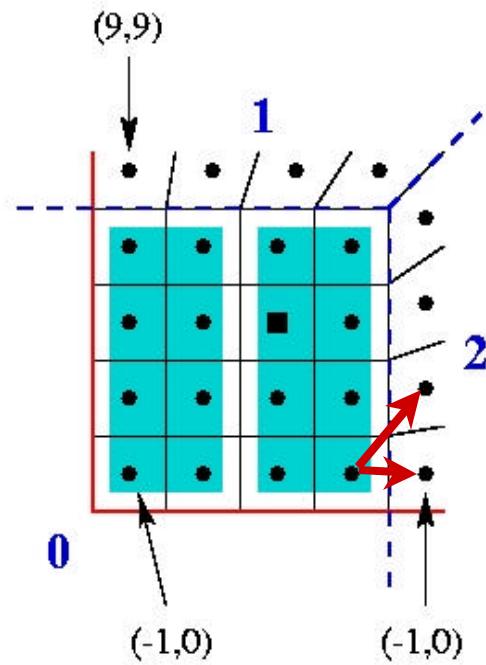
```
HYPRE_SStructGraph graph;
int index[9][2] = {{ 2,0},{2,1},...};
int to_indexes[9][2] = {{-1,0},{-1,1},...};

HYPRE_SStructGraphCreate(MPI_COMM_WORLD,
                        grid, &graph);

HYPRE_SStructGraphSetStencil(graph,
                            0, 0, stencil);

/* Add entries at part boundaries */
HYPRE_SStructGraphAddEntries(graph,
                            0, index[0], 0,
                            2, 2, &to_indexes[0], 0);
...
HYPRE_SStructGraphAssemble(graph);
```

A block-structured grid example : setting up the Matrix



$$\begin{bmatrix} S7 & S4 & S8 \\ S1 & \textcolor{red}{S0} & S2 \\ S5 & S3 & S6 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & \textcolor{red}{8} & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

```

int  sentries[2] = {0,3};
int  gentries[4] = {9,10,11,0};

HYPRE_SStructMatrixCreate(MPI_COMM_WORLD,
graph, &A);
HYPRE_SStructMatrixInitialize(A);
HYPRE_SStructMatrixSetBoxValues(A,
0, ilo[0], iup[0], 0,
2, sentries, values);

...
/* set values at non-stencil entries */
HYPRE_SStructMatrixSetValues(A,
0, index[0], 0, 2, gentries, values);

...
/* zero entries outside domain/part */
/* set bc's at domain boundaries, */

HYPRE_SStructMatrixAssemble(A);

```

Building different matrix/vector storage formats with the SStruct interface

- Efficient preconditioners often require specific matrix/vector storage schemes
- Between `Create()` and `Initialize()`, call:

```
HYPRE_SStructMatrixSetObjectType(A, HYPRE_PARCSR);
```

- After `Assemble()`, call:

```
HYPRE_SStructMatrixGetObject(A, &parcsr_A);
```
- Now, use the ParCSR matrix with compatible solvers such as BoomerAMG (algebraic multigrid)

Finite Element Interface (FEI)

- The FEI interface is designed for finite element discretizations on unstructured grids
- See the following for information on how to use it

R. L. Clay et al. An annotated reference guide to the Finite Element Interface (FEI) Specification, Version 1.0. Sandia National Laboratories, Livermore, CA, Technical Report SAND99-8229, 1999.

Linear-Algebraic System Interface (IJ)

- The IJ interface provides access to general sparse-matrix solvers, but not specialized solvers

```
HYPRE_IJMatrixCreate(MPI_COMM_WORLD, &A, M, N);
HYPRE_IJMatrixSetLocalStorageType(A, HYPRE_PARCSR);

HYPRE_IJMatrixInitialize(A);

/* set matrix coefficients a row at a time */
HYPRE_IJMatrixSetValues(A, num_values,
    row, cols, values);
...

HYPRE_IJMatrixAssemble(A);
parcsr_A = HYPRE_IJMatrixGetLocalStorage(A);
```

Several Solvers and Preconditioners are available in *hypre*

- Current solver availability via conceptual interfaces

Solvers	System Interfaces			
	Struct	SStruct	FEI	IJ
Jacobi	X			
SMG	X			
PFMG	X			
BoomerAMG	X	X	X	X
ParaSails	X	X	X	X
PILUT	X	X	X	X
PCG	X	X	X	X
GMRES	X	X	X	X

Setup and use of solvers is largely the same (see *Reference Manual* for details)

- **Create the solver**

```
HYPRE_SolverCreate(MPI_COMM_WORLD, &solver);
```

- **Set parameters**

```
HYPRE_SolverSetTol(solver, 1.0e-06);
```

- **Prepare to solve the system**

```
HYPRE_SolverSetup(solver, A, b, x);
```

- **Solve the system**

```
HYPRE_SolverSolve(solver, A, b, x);
```

- **Get solution info out via conceptual interface**

```
HYPRE_StructVectorGetValues(struct_x, index,  
values);
```

- **Destroy the solver**

```
HYPRE_SolverDestroy(solver);
```

Solver example: SMG-PCG

```
HYPRE_StructPCGCreate(MPI_COMM_WORLD, &solver);

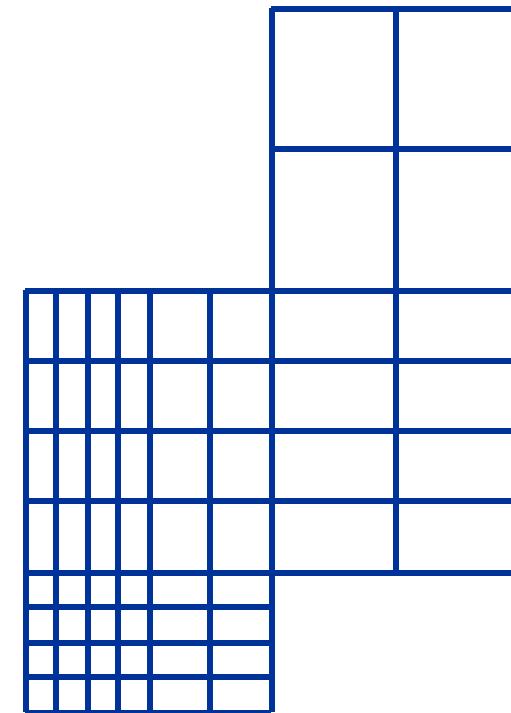
/* set stopping criteria */
HYPRE_StructPCGSetTol(solver, 1.0e-06);

/* define preconditioner (one symmetric V(1,1)-cycle) */
HYPRE_StructSMGCreate(MPI_COMM_WORLD, &precond);
HYPRE_StructSMGSetMaxIter(precond, 1);
HYPRE_StructSMGSetTol(precond, 0.0);
HYPRE_StructSMGSetZeroGuess(precond);
HYPRE_StructSMGSetNumPreRelax(precond, 1);
HYPRE_StructSMGSetNumPostRelax(precond, 1);

/* set preconditioner and solve */
HYPRE_StructPCGSetPrecond(solver,
    HYPRE_StructSMGSolve, HYPRE_StructSMGSetup, precond);
HYPRE_StructPCGSetup(solver, A, b, x);
HYPRE_StructPCGSolve(solver, A, b, x);
```

SMG and PFMG are semicoarsening multigrid methods for structured grids

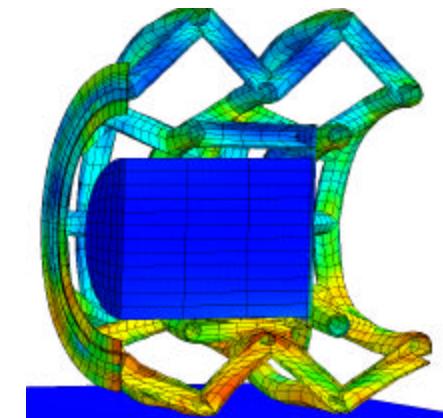
- Interface: Struct
 - Matrix Class: Struct
-
- SMG uses plane smoothing in 3D, where each plane “solve” is effected by one 2D V-cycle
 - SMG is very robust
 - PFMG uses simple pointwise smoothing, and is less robust



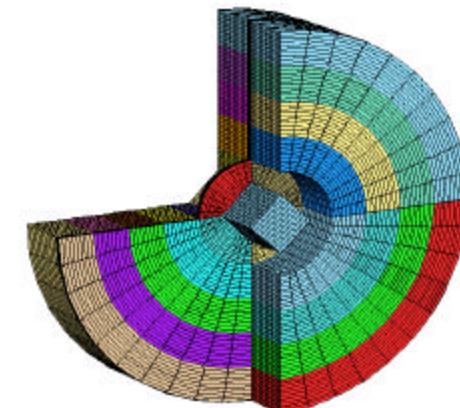
Our largest run (with PFMG-CG): 1B unknowns on 3150 processors of ASCI Red in 54 seconds

BoomerAMG is an algebraic multigrid method for unstructured grids

- Interface: `sStruct`, `FEI`, `IJ`
 - Matrix Class: `ParCSR`
-
- Originally developed as a general matrix method (i.e., assumed given A , x , and b only)
 - Uses simple pointwise relaxation
 - Automatically defines coarse “grids”



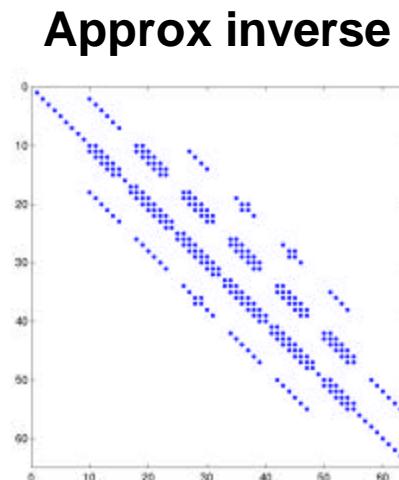
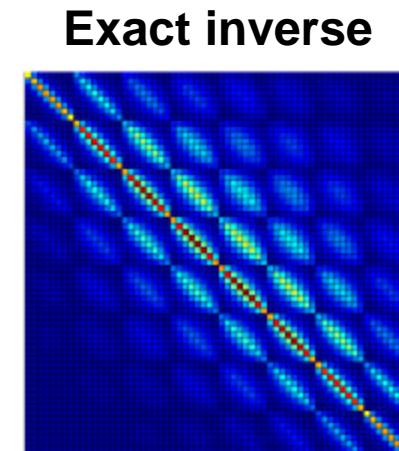
DYNA3D



PMESH

ParaSAILS is an approximate inverse method for sparse linear systems

- Interface: `sStruct`, `FEI`, `IJ`
 - Matrix Class: `ParCSR`
-
- Approximates the inverse of A by a sparse matrix M by minimizing the Frobenius norm of $I - AM$
 - Uses graph theory to predict good sparsity patterns for M



PILUT is an Incomplete LU method for sparse linear systems

- **Interface:** SStruct, FEI, IJ
 - **Matrix Class:** ParCSR
-
- **Uses thresholding drop strategy plus a mechanism to control the maximum size of the ILU factors**
 - **originally by Kumar and Karypis for T3D**
 - **now uses MPI and more coarse-grain parallelism**
 - **uses Schur-complement approach to parallelism**

Building the library

- Usually, *hypre* can be built by typing `configure` followed by `make`
- Configure supports several options (for usage information, type ‘`configure --help`’):
 - ‘`configure --enable-debug`’ - turn on debugging
 - ‘`configure --with-openmp`’ - use openmp
 - ‘`configure --with-CFLAGS=...`’ - set compiler flags

Calling *hypre* from Fortran

- C code:

```
HYPRE_IJMatrix A;  
int nvalues, row, *cols;  
double *values;  
  
HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values);
```

- Corresponding Fortran code:

```
integer*8 A  
integer nvalues, row, cols(MAX_NVALUES)  
double precision values(MAX_NVALUES)  
  
HYPRE_IJMatrixSetValues(A, nvalues, row, cols, values)
```

Reporting bugs

<http://www-casc.llnl.gov/bugs>

- Submit bugs, desired features, and documentation problems, or query the status of previous reports
- First time users must click on “Open a new Bugzilla account” under “User login account management”



This work was performed under the auspices of the U.S.
Department of Energy by Lawrence Livermore National
Laboratory under contract no. W-7405-Eng-48.